DESCRIPTION 531 Rec'd PCT 13 NOV 2001

Longest Matching Prefix Lookup

Field of Invention

Present invention is related to longest matching prefix lookup, in particular for determining, in a switching node, an output or forwarding route in response to a given search argument, such as an IP address.

Background of Invention

In modern communication networks, in particular the Internet, many users are connected to the network for exchanging addressed messages (or packets). Transmission links of the network are interconnected by switching centers or routing nodes (forwarding nodes) which, in response to the address of a received message (packet), forward this message to a selected one of the outgoing links. Thus, they have to route each packet by evaluating its address. This is also the case for message transmission in OSI, ATM, and other communication networks.

The problems of todays networks are (a) the great number of different users or addresses in the system, and (b) the tremendous number of messages (packets) to be transmitted. A forwarding node (routing center) in the Internet may have to dispatch about one million messages per second.

Another challenge is the provision of differently sized „domains" of addresses. Each of the addresses used may have a fixed total length (e.g. 32 bits); whole groups of addresses having a common „prefix" are assigned to the same domain and thus require the same routing, e.g. all addresses with prefix „4.22/16". One problem is that these domains and thus the prefixes to be considered during evaluation for routing in many systems (e.g. the Internet) have different sizes. Another problem is that there may be exceptions, i.e. the addresses of subdomains of a main domain may require different routings than the majority of addresses of the respective main domain. This requires „filtering out" those particular addresses during routing. For example, if most of the users with addresses of the group 4.22.xxx.xxx (4.22/16) are located in the same area, the addresses of subdomain 4.22.48.xxx ..... 4.22.55.xxx (4.22.48/20) and

the addresses of another, even smaller subdomain 4.22.17.192 ..... 4.22.17.255 (4.22.17.192/26) may require routing through other links because the respective recipients (users) are located in another region.

This leads to the technique of longest matching prefix (or best matching prefix) routing which requires a respective evaluation of addresses in each forwarding node (routing center).

Several techniques were developed for effecting longest matching prefix routing. Most are based on the use of hierarchical tree like structures. In such trees, information on the output link or route to be taken (e.g. a next-hop pointer) is stored in a leaf (exit point / end point) of the tree, while the path through the tree is determined by the addresses evaluated. To find the correct path, i.e. to determine the branching (in each node of the tree), each address (or portions thereof) are either (a) compared in each node to address information stored there, or (b) used to directly (or indirectly) control the branching, i.e. select the next partial path to be taken.

In any case, no matter which technique is used, the following must be achieved:

(1) address evaluation and route determination must be done in a time interval that corresponds to the average time available in the worst case (i.e. in high traffic situations) for each message (packet);

(2) each address evaluation should require as few storage accesses or processing operations as possible; and

(3) the storage space required for the tables (data bases) should be as small as possible.

While a solution may be a compromise between (2) an (3), the first criterion must be met in any case to avoid undue delays or even clogging of a routing node.

A further aspect for designing such routing sytems is its flexibility, i.e. the possibility to adapt the structure to changing domain sizes and address distributions in the system.

Known Approaches

Several methods and systems were suggested for finding the longest matching prefix of a given search argument, and to produce a corresponding (assigned) pointer or output identification.

A paper "Faster IP Lookups using Controlled Prefix Expansion" by V.Srinivasan and G.Varghese, published in Perform.Eval.Rev. (USA) Vol.26, No.1, June 1998 pp.1-10, describes a method in which only a selected number of different prefix lengths is allowed, and in which given prefixes of addresses are expanded in length if they do not have one of the selected lengths. The search database is designed to have several levels of tables, each level corresponding to one of the prefix lengths. Then, an incremental search can be made, using on each level only a portion of the given search address. The table entries on each level contain pointers to the next table to be searched with the next portion of the search address.

This approach allows a reduction in storage space required and in the number of operations necessary. However, it does not allow to directly compare a segment of the search address to a test value to enable an immediate decision when a particular prefix is present. Furthermore, the suggested method requires the expansion of prefixes instead of directly using all prefixes in their given length. Due to the given structure based on preselected lengths, changes in the selected lengths require an adaptation of the whole structure of the database. Also, though different sizes of search address segments can be used on the different levels, only one segment length is possible per level.

In U.S.Patent 5,781,772 (Wilkinson et al.) "Compressed Prefix Matching Database Searching" a technique is disclosed for reducing the length of searches and the storage requirements, by pointer compression (i.e. eliminating pointers to NIL nodes), and by path compression (eliminating nodes contributing no additional relevant information, and storing the associated prefix digit or digits as path digit string which is compared to respective portions of the given address). However, the method provides only a sectioning of search addresses into fixed length segments (or multiples thereof), and it does not appear to provide a combination of indexed pointer addressing of node entries and comparison of selectable-length address parts to stored values.

In a paper by T.Harbaum, D.Meier, M.Zitterbart and D.Brokelmann "Hardware-Assist for IPv6 Routing Table Lookup", published in Proc. Internat.Symposium on Broadband Networks (SYBEN'98), Zurich/Switzerland May 1998, a binary tree search method is described for finding a best match for different-length prefixes of IP addresses. The method allows

comparison of variable-length portions of address prefixes to stored values to enable a compacted search data base. However, only binary search decisions are provided, and therefore in many cases multiple comparisons and decisons are required for the same output route.

Another approach for handling different prefix lengths in a matching search is described in a paper by P.Gupta, S.Lin and N.McKeown "Routing Lookups in Hardware at Memory Access Speeds", published in Proc. IEEE INFOCOM'98, 1998, Vol.3 pp.240-247. The disclosed method provides, for 32-bit addresses, two tables. One is addressed by the first 24 bits and has 2^{24} entries, one for each of the 24-bit prefixes. If the relevant prefix has less than or at most 24 bits, then the respective entry contains the pointer to the associated output or link. If the relevant prefix is longer, then the entry contains a pointer to a second table which is then addressed by combining the pointer and the remaining eight bits of the prefix (after expansion to 32 bits). The entries of the second table contain the pointers to the outputs/links for the longer prefixes. An additional intermediate table allows to reduce the size of the second table in specific situations (for prefixes shorter than 32 bits). This method requires only one, two, or three storage accesses per address match, but needs a large storage capacity, and addition or deletion of a prefix necessitates the changing of many entries.

U.S.Patent 5,386,413 (McAuley et al.) "Fast Multilevel Hierarchical Routing Table Lookup Using Content Addressable Memory" allows fast simultaneous lookup for different prefix segments of a given address, but requires a content-addressable storage.

None of these disclosed methods and systems allows to use search argument segments of selectable length partially as testing values and partially as addressing indexes in a flexible combination, for finding longest matching prefixes in a multiple-table structure having entries with varying, different access criteria.

Objects of the Invention

It is the object of the invention to devise a method and data structure design for finding a longest matching prefix for a given input address or search argument, in a multiple-level arrangement of stored tables, which requires only a few lookup operations per search, has

moderate storage requirements, and allows a flexible adaptation of the stored data structure to the distribution of used prefixes in a system.

The invention for achieving this object is defined in the claims. It allows, in particular, the use of variable-length segments of the given search argument as either test values or addressing indexes, and also a combination of testing followed by indexed accessing with such variable-length segments, during the lookup procedure for detection of longest matching prefixes and their assigned output indications.

With the invention, it is possible to optimally adapt the used node table data structure and the search procedure for longest matching prefixes, to the actual distribution of prefixes over the whole address space, i.e. to use different structures depending on the fact whether there is a sparsed or clustered distribution.

Depending on the frequency of access of particular prefixes, the search path for the most frequently used prefixes can be optimized by giving them a treatment that shortens the search, whereas for rarely used prefixes a somewhat lengthier search can be allowed to gain other advantages, e.g. saving of storage space. The invention enables, due to the possible direct comparison of a long address segment to a stored test value, a fast completion of the search for specific prefixes. Furthermore, when using the invention it is possible with a single lookup operation to select between several (more than two) choices.

The invention also provides for two-stage addressing of entries, i.e. a node table is selected by a table base pointer, and the entry selection within the table can be done by an offset in response to a test result or an index operation, so that fast and flexible moving between entries is possible.

The invention also allows the use of constant-size entries in the node tables wich simplifies the data structure and access to as well as updating of entries.

Finally, providing a reuse function in a system using the invention offers the possibility to first make a test on a long segment of the search argument and then, depending on the result, to do a subsequent finer evaluation in smaller steps.

Embodiment examples of the invention are described in the following with reference to drawings.

List of Drawings

Fig.1 shows a first, general embodiment example of the invention.

Fig.2 is a flow diagram for the embodiment of Fig.1 showing the different possible steps in evaluating a search argument.

Fig.3 illustrates a second example of an embodiment of the invention, in which test values are not separately stored but within respective table entries.

Fig.4 shows a third example of an embodiment in which portions of search arguments (IP addresses) which were already evaluated can be reused again.

Fig.5 (5A and 5B) illustrates the generation of a node table structure adapted to a given set of different prefix values.

Fig.6 (6A and 6B) shows the generation of an optimized node table structure on the basis of the case shown in Fig.5.

Fig.7 illustrates the selection of test values and indexes and the resulting branching scheme for the table structure generation procedure of Fig.5.

Fig.8 illustrates the selection of test values and indexes and the resulting branching scheme for the table structure generation procedure of Fig.6.

Detailed Description

In the following, three examples of an embodiment of the invention are described (one basic, two variations). In a final section, an example of an optimized data structure is given, showing how such a data structure can be generated and optimized for a given set of prefixes of search arguments.

It is important to note that selectable portions of the search argument of variable length are used either as index (value) for selecting/accessing node table entries, or as test arguments to be compared to stored test values, but that both possibilities are used in combination to adapt the search procedure to the existing or expected distribution of the occurring prefixes. This allows an optimization with respect to either processing speed, number of operations, or the storage requirements for the search (lookup) data base. Furthermore, for accessing any entry in the search data base, full addressing is not required. Rather, a table base pointer is used for selecting the respective node table, and access within the table can then be effected using an offset value (from the base address of the table), or the result of a test.

Following terms and abbreviations are used in the subsequent description:

* search argument: an IP address or packet address or keyword etc.
* node table: a table of entries each containing search information or an output route information (the node tables are arranged in multiple successive levels)
* index: a value used for determining the offset from the base address when accessing node table entries
* offset value: a value indicating the distance between the start address of a node table and the location of an entry to be accessed
* table base pointer: a pointer indicating the start (address) of a node table = PTR
* end indicator: an indication in a last entry in a search path = END; either an output indicator or a stop indicator not leading to an output (may be an empty entry = NIL)

* output indicator: a pointer to the selected output route (e.g. a Next Hop Pointer = NHP) *

test value: a bit sequence, stored in a table entry, to be compared to a portion of a search argument = TV

* test result: the result of the comparison between a test value and a test argument (selected portion of search argument); this is a binary value (match / no match, or 1 / 0)

First (Basic) Example of an Embodiment

A basic embodiment of a search (lookup) data structure according to the invention is schematically shown in Fig.1; the description is supported by the flow diagram of Fig.2. Selected node tables (only those which are used in the example) are shown on different levels, together with the respectively marked IP address (= search argument).

In the first example, the following four different (kinds of) entries occur in the node tables:

-- *Basic Entries:* 1) F1 + F2 / CNT1 / CNT2 / PTR

2) F1 + F2 / CNT / PTR

-- *Additonal Entries*: 3) F1 + F2 / END

= Output indicator (NHP) OR Stop indicator (NIL)

4) TV

= Test value (TV) only

In the basic entries, F1 + F2 indicate the operation to be performed (comparison to a test value / single indexed accessing of a table entry / combination of both); CNT or CNT1 and CNT2 are selection information indicating the number of bits to be taken from the search argument (as test argument or index) for the requested operation(s); and PTR is a table base pointer indicating the start address of a node table in which the test value is contained or in which the next entry (entries) are to be accessed.

An additional entry contains: Either flags F1 and F2 (indicating a last entry), with an output indicator (next hop pointer NHP), or with a stop indicator (NIL) indicating that the search is finished (no output route available); or it contains just a single test value (TV).

Flag combinations used in this and the other examples have the following meaning:

F1	F2	
0	0	Last entry (ouput indicator or stop indicator)
1	0	Test to be made (one CNT field)
0	1	Index to be used for entry selection (one CNT field)
1	1	Test followed by Index operation (two CNT fields: CNT1, CNT2)

The set of prefixes and associated next hop information used for this (and also the other two examples) is the following:

	Prefix	Bit vector	Next hop pointer
A	4.22.17/24	0000 0100 0001 0110 0001 0001	1
B	4.22.17.0/26	0000 0100 0001 0110 0001 0001 00	2
C	4.22.17.64/26	0000 0100 0001 0110 0001 0001 01	3
D	4.22.17.243/32	0000 0100 0001 0110 0001 0001 1111 0011	4

The basic method described in this example is characterized by:

- the test value is stored at the beginning of a node table at the next level
- each test has a binary result: a "match" or a "no match"
- no reuse flag; each portion of the IP address is used at most only once

The procedure in the basic embodiment is as follows:

Step 1:

The first 16 bits of the IP address (search argument) are selected as an initial segment and used as index into a table at level 1 consisting of $2^{16} = 65'536$ entries. The entry at index 1046 is selected when the IP address starts with (i.e., has a prefix equal to) 4.22/16.

Step 2:

This entry at index 1046 has flag combination „11" which means that first a test and then - conditionally - an index addressing operation must be made. Based on the selected entry, a segment consisting of 8 successive bits (= CNT1) within the IP address is selected and tested

against a test value 17. This test value is found in the first (lowest) entry of the node table at the next level 2, which is identified by the table base pointer in entry 1046.

If the test results in "no match" then the "no match" entry is selected in the table at level 2 which is a last entry containing a stop indicator "-" meaning that no result (output) is specified for the given prefix.

If the test results in a "match" then a segment consisting of 2 successive bits (= CNT 2) within the IP address is selected and used as an index into the "match" part of the selected table at level 2. This part starts above the two "fixed" entries for the test value and for the no-match result, and it comprises 4 entries to be accessed using an offset value which is derived from the index. (In the figure, the index values 0...3 are shown; they would result in offset values 2...5, meaning entries 2...5 in the respective node table of which the entry 0 containing the test value is identified by the table base pointer).

-- An IP address with prefix B (index bits 00) will result in the selection of the entry with index 0 which is a last entry (flags = 00) containing next hop pointer "2".

-- An IP address with prefix C (index bits 01) will result in the selection of the entry with index 1 which is also a last entry (flags = 00), containing next hop pointer "3".

-- An IP address with the first 26 bits equal to those of prefix D (presently used index bits 11) will result in the selection of the entry with index 3 which requires an additional test (flags = 10) which is described below as step 3.

-- All other IP addresses which had a "match" result in the previous test (these are the remaining matching addresses, having index bits 10) have prefix A and will result in the selection of the entry with index 2 in the match part of the node table at level 2, which is a last entry (flags = 00) containing next hop pointer "1".

Step 3:

Based on the entry at index 3 in the match part of the table at level 2, a segment consisting of 6 successive bits (CNT = 6) within the IP address is selected and tested against a test value 51 (which is found in the lowest entry of the table at level 3, which is identified by the table base pointer found in the respective entry of the last used node table at level 2).

-- For an IP address with prefix D (having a bit sequence 110011 = 51 in the respective address positions) the test results in a "match" leading to the "match" entry in the selected node table at level 3, wich is a last entry (flags = 00) containing next hop pointer "4".

-- For IP addresses which do not have a prefix D, the test against value 51 results in a "no match" condition, thus selecting the "no match" entry in the selected node table at level 3, which is also a last entry (flags = 00), containing next hop pointer "1". These addresses have a prefix A (all remaining addresses with prefix A, i.e. other than those which would have been handled already in step 2).

The flow diagram of Fig.2 illustrates how the entries are handled in response to the flags (bit pair in the first positions) contained in each basic entry. This scheme corresponds to the above description.

Second Embodiment Example

A second embodiment example is shown in Fig.3. This is very similar to the first example, with the difference that the test values to be used are not kept in separate entries (of a node table at the next level) but are rather directly contained in the respective entry requiring the test to be made. Therefore, the node table entries must have an additional field which is shown in the modified set below:

-- *Basic Entries:* 1) F1 + F2 / CNT1 / CNT2 / TV / PTR

2) F1 + F2 / CNT / TV /PTR

-- *Additonal Entries:* 3) F1 + F2 / END

= Output indicator (NHP) --or-- Stop indicator (NIL)

It is clear that this modification is useful when the test values are usually short (a few bits only) whereas the solution of the first example is better suited when the search data base will contain many long test values. This second solution saves the storage space for the separate test value entries, and further avoids the additional access operations for fetching the test values.

It appears that on the basis of Fig.3 and with the description given for the first example, no further detailed description is necessary here for the second example.

Third Embodiment Example

A furher, third embodiment example is shown in Fig.4. This is also somewhat similar to the first example. The same set of prefixes A...D is also used here. However, in contrast to the first (and second) example, the present example provides for the reuse of portions (bit groups) of the search argument (IP address) which were already evaluated in a previous step. In the first (and second) example, always subsequent (new) groups of bits from the search argument are used for test operations or as indexes in accessing node table entries.

The reuse of a selected portion (bit group) from an IP address (search argument) may be useful in the following situation: When a set of prefixes occurring in an application contains one or a few very specific (long) prefixes, one could first make a test of a long segment of this specific prefix against a respective test value to get an immediate result if a match is detected. Otherwise (if the result is "no match" indicating that this specific prefix is not present), a portion of the already used long prefix segment is then used in another operation to distinguish other, more similar prefixes from each other in several subsequent operations. This may result in faster search (IP address evaluation) operation in systems where such specific, isolated prefixes occur.

The set of node table entries to be used for this example is similar to that of example 1, except that that the basic entries contain an additional "Reuse" flag.

-- *Basic Entries:*

1) F1 + F2 / R / CNT1 / CNT2 / PTR

2) F1 + F2 / R / CNT / PTR

-- *Additonal Entries*:

3) F1 + F2 / END

= Output indicator (NHP) --or-- Stop indicator (NIL)

4) TV

= Test value (TV) only

The procedure in this third example according to Fig.4 is as follows (the set of prefixes A...D is the same as in the first example, but the stored data structure in the node tables is of course different):

The R (Reuse) flag in a basic entry is set when a certain segment of an IP address is used for the first time and has to be reused again (under certain conditions depending on a test result).

Step 1: (same as in the basic method of example 1)

The first 16 bits of the IP address are selected as an initial segment and used as an index into a table at level 1 consisting of $2^{16} = 65'536$ addresses. The entry at index 1046 is selected when the IP address starts with (i.e. has a prefix equal to) 4.22/16.

Step 2:

Based on the selected entry at index 1046 in the table at level 1, a segment consisting of 16 successive bits (CNT) within the IP address is selected and tested against a test value 17.243 (bit sequence 0001 0001 1111 0011). The reuse flag (R) is set which means that at least a portion of the selected segment may be used again. The test value is found in the lowest position of a node table at level 2, which is identified by the table base pointer in the entry 1046.

If the test results in a "match", then the "match" entry is accessed in the selected node table at level 2. This is a last entry (flags = 00) containing the next hop pointer "4". This is the case for IP addresses with prefix D.

If the test results in "no match", then the "no match" entry is accessed in the selected node table at level 2. This entry specifies a combined test/index operation (flags = 11) which is described in step 3 below, and is selected for all IP addresses that start with 4.22 (bit sequence 0000 0100 0001 0110) but do not have the prefix D.

Step 3:

Based on the "no match" entry in the node table at level 2 (specifying first a test operation based on CNT1 = 8), a new segment of 8 successive bits is selected from the segment of 16

bits out of the IP address, which was used in the previous operation. This reuse was determined by the R flag set to 1 in the entry 1046 at level 1. The reuse Flag R in the "no match" entry of level 2 is set to 0 since the now selected segment of 8 bits will not be reused again.

If the test results in "no match" then the "no match" entry is selected in a node table at level 3 specified by the table base pointer in the last accessed entry. The newly selected entry is a last entry (flags = 00) containing a stop indicator "-" (meaning that no output route is specified for IP addresses having a prefix other than 4.22.17 and that the search is finished).

If the test results in a "match", then the indexing operation specified in the"no match" entry of the table at level 2 is effected, using CNT2 = 2. Thus, a segment of 2 successive bits (following the 8 address bits selected in the previous step for the test operation) is selected from the IP address and used as an index for accessing an entry in the "match" part of the selected node table at level 3. This part has four entries of which one is identified by an offset value deducted from the recently selected two index bits.

-- An IP address with prefix B (index bits 00) will result in the selection of the entry in the "match" part of the table at level 3, associated with index 0. This is a last entry containing next hop pointer "2".

-- An IP address with prefix C (index bits 01) will result in the selection of the entry in the "match" part of the table at level 3, which is associated with index 1. This is also a last entry, containing next hop pointer "3".

-- IP addresses that do not have a prefix B, C, or D (but start with the sequence 4.22.17) must have index bits 10 or 11 at this stage and thus will result in the selection of one of the two remaining entries in the "match" section of the table at level 3, i.e. those associated with index 2 or 3. They both contain next hop pointer "1" which corresponds to prefix A.

Conclusion: When comparing the basic solution of example 1 with the variation of example 3, one can see that they both lead to the same result: IP addresses with prefixes A, B, C, or D will be correctly directed to output routes corresponding to next hop pointer 1, 2, 3, or 4, respectively. All other IP addresses result in a termination of the search. - However, the basic

solution (example 1) may be more advantageous in situations where addresses with prefixes B and C occur more frequently because lookups are faster for these prefixes. In contrast, the variation of example 3 is more effective in situations where addresses with prefix D occur more frequently because of the shorter lookup time for this prefix D.

Remark on entry selection: As was mentioned in the beginning, and was shown in the three examples, the selection of an entry in a node table that was identified by a table base pointer, can be effected in different ways:

When a test was made (against a stored test value), there is a binary result, "match" or "no-match". Then there are two possibilities:

a) Each of the two results causes the selection of one specific entry in the node table, always associated with that result (c.f. Fig.4, transition from level 1 to level 2).

b) One of the two results ("no match") causes the selection of one specific entry in the node table, always associated with that result. The other of the two results ("match") causes the additional selection of an index value which in turn is then used to select one of two (or four, or eight) entries in a special portion ("match") of the respective node table (cf. transition from level 2 to level 3 in Fig.4).

Of course, direct addressing of an entry (in a table not containing entries fixedly associated to the result of a test operation) is also possible using an index value. This is the case e.g. in the initial step on level 1 of Fig.1 and Fig.4.

These different entry selection possiblities allow a very flexible adaptation of the search procedure and the search data base to the expected distribution of prefixes.

Generation of Optimized Data Structure for Prefix Lookup

In the three exmples described above, the different data structure elements and also basic and combined operations, which are provided by the invented lookup procedure for longest matching prefixes, were presented and explained. As was mentioned, they allow an optimization of the lookup process with respect to operation speed or storage and processor requirements. The search data base must be so structured and the node table entries so arranged that they best fit the expected distribution of occurring prefixes.

An example for such optimization is illustrated in Fig.5 (5A and 5B) and Fig.6 (6A and 6B) and also in Fig.7 and Fig.8 and explained in the following.

The type of entries used and the designation of fields are the same as those in the first example presented in the beginning of this detailed description.

The set of prefixes assumed for present optimization example is shown below. This is of course a reduced set of different prefixes which is much smaller than those really occurring. But it should be sufficient for illustrating the principle.

	Prefix	Bit vector	Next hop pointer
A	9.20/14	0000 1001 0001 01	1
B	9.20.157/24	0000 1001 0001 0100 1001 1101	2
C	9.20.136.16/28	0000 1001 0001 0100 1000 1000 0001	3
D	9.20.155/24	0000 1001 0001 0100 1001 1011	4

Fig.5 shows, for this set of prefixes, a first data structure of the multiple-level node table arrangement. This data structure is created in the following way: If prefixes share a common most significant part (leading prefix), then this part is selected and stored as a test value for a first test operation. Then, the next bit, i.e. the first bit that is different, is determined to be used as index to select between two separate search paths. In the same way, along a given search path, the common most significant part of the remaining portion of the prefixes in that search path is selected and stored as a test value for test operations. Then, the next bit, i.e. again the first bit that is different in the remaining portion of the prefixes on this search path, is determined to be used as an index to select between two succeeding paths, and so on. For a prefix that is a portion of another, longer prefix (and thus does not have a different bit but is only shorter), no index is used but the test value covers the bits up to the end of the shorter prefix that is a portion of the other prefix.

The selection of test values and indexes during this process and the resulting branching scheme are illustrated in Fig.7. In this figure, a rectangular box indicates a test operation between an address segment and the bit sequence (test value) shown in the box. These test values will be stored in the respective node table entries. A double circle indicates an indexing

operation, the number of "b"s representing the number of successive bits of the address used for indexing, and their actual values being shown on the outgoing lines. When the output of a test or index operation reaches an entry with an output indicator (next hop pointer) for a prefix, this is indicated by the encircled prefix letter at that output.

Details of procedure: Prefix A is common to the prefixes B, C, and D. The data structure (Fig.5) therefore starts with a test operation covering up to the end of prefix A ('0000 1001 0001 01b'). The remaining portions of prefixes B, C, and D share the common part '00 100b'. This becomes the test value stored in the table at level 2. In case of a negative test result, the prefix A was the longest matching prefix for respective search arguments, and the corresponding next hop pointer "1" is stored in the "no match" entry in the table at level 2.

Thereafter, a one bit index is used to select between two separate paths. Only prefix C will 'go' along the search path with an index value '0'. The remaining bits of prefix C are included as a test value in the lower table at level 3. A positive test result means that prefix C is the longets matching prefix and therefore the next hop pointer "3" corresponding to prefix C is stored in the "match" entry. A negative test result means that prefix A is the longest matching prefix for all respective search arguments and therefore the next hop pointer "1" corresponding to a prefix A is stored in the "no match" entry. The same approach is used along the other search path for prefixes B and D.

The data structure shown in Fig.5 (5A and 5B) which was generated according to the above rules, can be further optimized to improve the lookup speed for certain prefixes by increasing the sizes of the indexes that are used along the search path for those specific prefixes and adapting the remaining parts of the succeeding search paths (for example by removing bits that now have become part of an index, from a test value in the succeeding test operation). Fig.6 (6A and 6B) shows an optimized data structure in which the lookup speed is incrased for the prefixes B and D by increasing the size (bit number) of the index from 1 to 3 (CNT2 field in the "match" entry at index "1" within the table at level 2 of Fig.6B is given a value of 3).

Fig.8 shows the selection of test values and indexes during generation of an optimized search data structure as shown in Fig.6, and the resulting branching scheme. A comparison between

Fig.7 and Fig.8 shows how at any particular place of the searching tree, the design can be changed for increasing the search speed by reducing the number of required operations in exchange for an increased number of entries in the table storage, or vice versa.